Binary Decision Diagrams

Lecture #13 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

December 11, 2006

Ordered Binary Decision Diagram

share equivalent expressions [Akers 76, Lee 59]

- Binary decision diagram (OBDD) is a directed graph over $\langle X, < \rangle$ with:
  - each leaf v is labeled with a boolean value $\mathit{val}(v) \in \{ 0, 1 \}$
  - non-leaf v is labeled by a boolean variable $\mathit{Var}(v) \in X$
  - such that for each non-leaf v and vertex w:

$$w \in \{ \mathit{left}(v), \mathit{right}(v) \} \;\Rightarrow\; (\mathit{Var}(v) < \mathit{Var}(w) \;\vee\; w \text{ is a leaf})$$

$\Rightarrow$ An OBDD is acyclic

- f_{B} for OBDD B is obtained as for BDTs

Reduced OBDDs

OBDD B over $\langle X, < \rangle$ is called *reduced* iff:

1. for each leaf v, w: $(\mathit{val}(v) = \mathit{val}(w)) \;\Rightarrow\; v = w$

   $\Rightarrow$ identical terminal vertices are forbidden

2. for each non-leaf v: $\mathit{left}(v) \neq \mathit{right}(v)$

   $\Rightarrow$ non-leafs may not have identical children

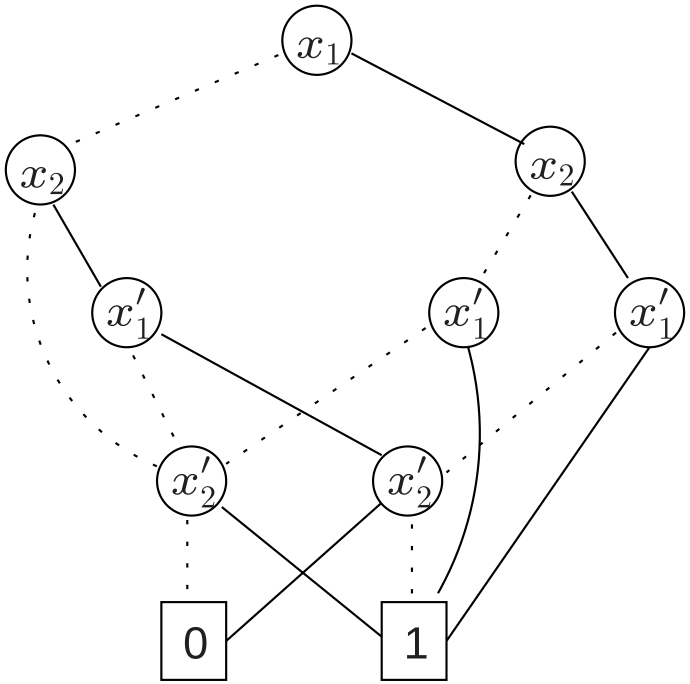
3. for each non-leaf v, w:

$$(\mathit{Var}(v) = \mathit{Var}(w) \;\wedge\; \mathit{right}(v) \cong \mathit{right}(w) \;\wedge\; \mathit{left}(v) \cong \mathit{left}(w)) \;\Rightarrow\; v = w$$

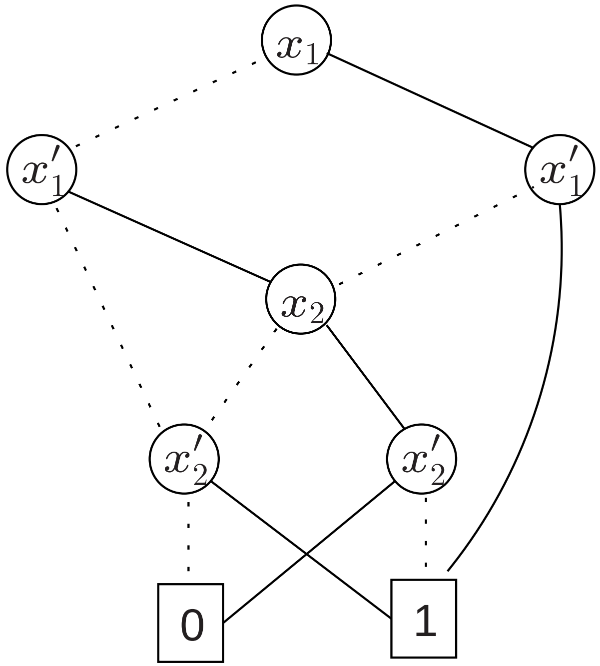
   $\Rightarrow$ vertices may not have isomorphic sub-dags

this is what is mostly called BDD; in fact it is an ROBDD!

Transition relation as an ROBDD



(a) ordering $x_1 < x_2 < x_1' < x_2'$



(b) ordering $x_1 <' x_1' <' x_2 <' x_2'$

Shannon expansion

- Each boolean function $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ can be written as:

$$f(x_1, \ldots, x_n) = (x_i \ \wedge \ f[x_i := 1]) \ \vee \ (\neg x_i \ \wedge \ f[x_i := 0])$$

  - where $f[x_i := c]$ stands for $f(x_1, \ldots, x_{i-1}, c, x_{i+1}, \ldots, x_n)$

- The boolean function $f_{\mathsf{B}}(v)$ represented by vertex v in BDT B is:
  - for v a leaf: $f_{\mathsf{B}}(v) = \mathit{val}(v)$
  - otherwise:

$$f_{\mathsf{B}}(v) = (\mathit{Var}(v) \ \wedge \ f_{\mathsf{B}}(\mathit{right}(v))) \ \vee \ (\neg \mathit{Var}(v) \ \wedge \ f_{\mathsf{B}}(\mathit{left}(v)))$$

- $f_{\mathsf{B}} = f_{\mathsf{B}}(v)$ where v is the root of B

ROBDDs are canonical

[Fortune, Hopcroft & Schmidt, 1978]

For ROBDDs B and B′ over $\langle X, < \rangle$ we have:

$(f_{\mathsf{B}} = f_{\mathsf{B}'})$ implies B and B′ are isomorphic

$\Rightarrow$ *for a fixed variable ordering, any boolean function can be uniquely represented by an ROBDD (up to isomorphism)*

The importance of canonicity

- Absence of redundant vertices
  - if f_{B} does not depend on x_i, ROBDD B does not contain an x_i vertex
- Test for equivalence: $f(x_1, \ldots, x_n) \equiv g(x_1, \ldots, x_n)$?
  - generate ROBDDs B_f and B_g, and check isomorphism
- Test for validity: $f(x_1, \ldots, x_n) = 1$?
  - generate ROBDD B_f and check whether it only consists of a 1-leaf
- Test for implication: $f(x_1, \ldots, x_n) \rightarrow g(x_1, \ldots, x_n)$?
  - generate ROBDD $\neg \mathrm{B}_f \vee \mathrm{B}_g$ and check if it just consist of a 1-leaf
- Test for satisfiability
  - f is satisfiable if and only if B_f is not just the 1-leaf

Variable ordering

- The size of the ROBDD depends on the variable ordering
- For some functions, very compact ROBDDs may be obtained
  - e.g., the even parity function
- Some boolean functions have linear and exponential ROBDDs
  - e.g., the addition function, or the stable function
- Some boolean functions only have polynomial ROBDDs
  - this holds, e.g., for symmetric functions (see next)
  - examples $f(\ldots) = x_1 \oplus \ldots \oplus x_n$, or $f(\ldots) = 1$ iff $\geqslant k$ variables x_i are true
- Some boolean functions only have exponential ROBDDs
  - this holds, e.g., for the multiplication function, cf. (Bryant, 1986)

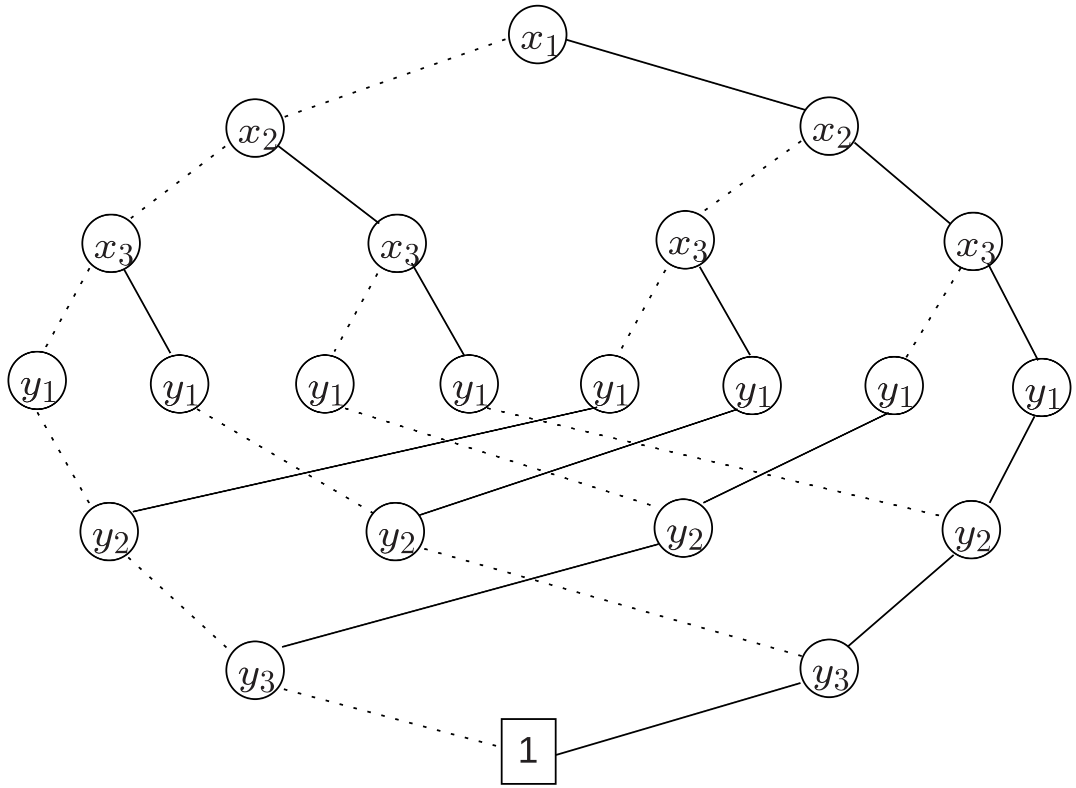
The even parity function

$f_{even}(x_1, \ldots, x_n) = 1$ iff the number of variables x_i with value 1 is even

truth table or propositional formula for f_{even} has exponential size

but an ROBDD of linear size is possible

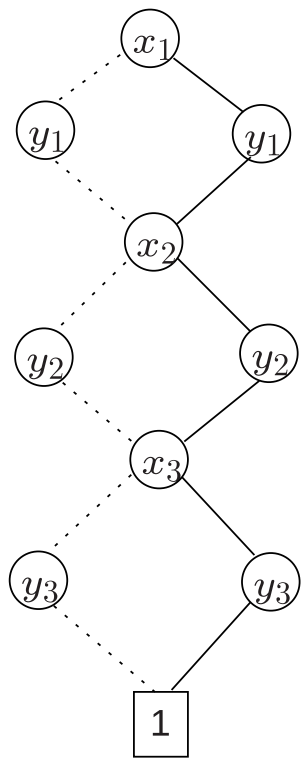
The function stable with exponential ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \ \wedge \ \ldots \ \wedge \ (x_n \leftrightarrow y_n)$

has $3 \cdot 2^n - 1$ vertices under ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$

The function stable with linear ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \ \wedge \ \ldots \ \wedge \ (x_n \leftrightarrow y_n)$

has $3 \cdot n + 2$ vertices under ordering $x_1 < y_1 < \ldots < x_n < y_n$

Symmetric function $(n=4)$

symmetric boolean functions have ROBDDs of size in $\mathcal{O}(n^2)$

The multiplication function

- Consider two n-bit integers
  - let $b_{n-1}b_{n-2}\ldots b_0$ and $c_{n-1}c_{n-2}\ldots c_0$
  - where b_{n-1} is the most significant bit, and b_0 the least significant bit
- Multiplication yields a $2n$-bit integer
  - the ROBDD $\mathrm{B}_{f_{n-1}}$ has at least 1.09^n vertices
  - where f_{n-1} denotes the the $(n-1)$-st output bit of the multiplication

Optimal variable ordering

- The size of ROBDDs is dependent on the variable ordering
- Is it possible to determine $<$ such that the ROBDD has minimal size?
  - the optimal variable ordering problem for ROBDDs is NP-complete
  - polynomial reduction from the 3SAT problem (Bollig & Wegener, 1996)
- There are many boolean functions with large ROBDDs
  - for almost all boolean functions the minimal size is in $\Omega(\frac{2^n}{n})$
- How to deal with this problem in practice?
  - guess a variable ordering in advance
  - rearrange the variable ordering during the manipulations of ROBDDs
  - not necessary to test all $n!$ orderings, best algorithm in $\mathcal{O}(3^n \cdot n^2)$

Variable swapping

Sifting algorithm

(Rudell, 1993)

Dynamic variable ordering using variable swapping:

1. Select a variable x_i

2. By successive swapping of x_i, determine $|B|$ at any position for x_i

3. Shift x_i to its optimal position

4. Go back to the first step until no improvement is made

- Characteristics:
  - a variable may change position several times during a single sifting iteration
  - often yields a local optimum, but works well in practice

Transition systems as boolean functions

- Assume each state is uniquely labeled
  - no restriction: if needed extend *AP* and label states uniquely
- Assume a fixed total order on propositions: $a_1 < a_2 < \ldots < a_K$
- Represent a state by a *boolean function*
  - over the boolean variables x_1 through x_K such that

$$[\![\, s \,]\!] \;=\; x_1^* \;\wedge\; x_2^* \;\wedge\; \ldots \;\wedge\; x_K^*$$

  - where the literal x_i^* equals x_i if $a_i \in L(s)$, and $\neg\, x_i$ otherwise
- Represent I and $\rightarrow$ by their characteristic (boolean) functions
  - e.g., $f_{\rightarrow}([\![\, s \,]\!], [\![\, \alpha \,]\!], [\![\, t \,]\!]) = 1$ if and only if $s \xrightarrow{\alpha} t$

Interleaved variable ordering

- Which variable ordering to use for transition relations?
- The *interleaved variable ordering*:
  - for encodings $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ of state s and t respectively:

$$x_1 < y_1 < x_2 < y_2 < \ldots < x_n < y_n$$

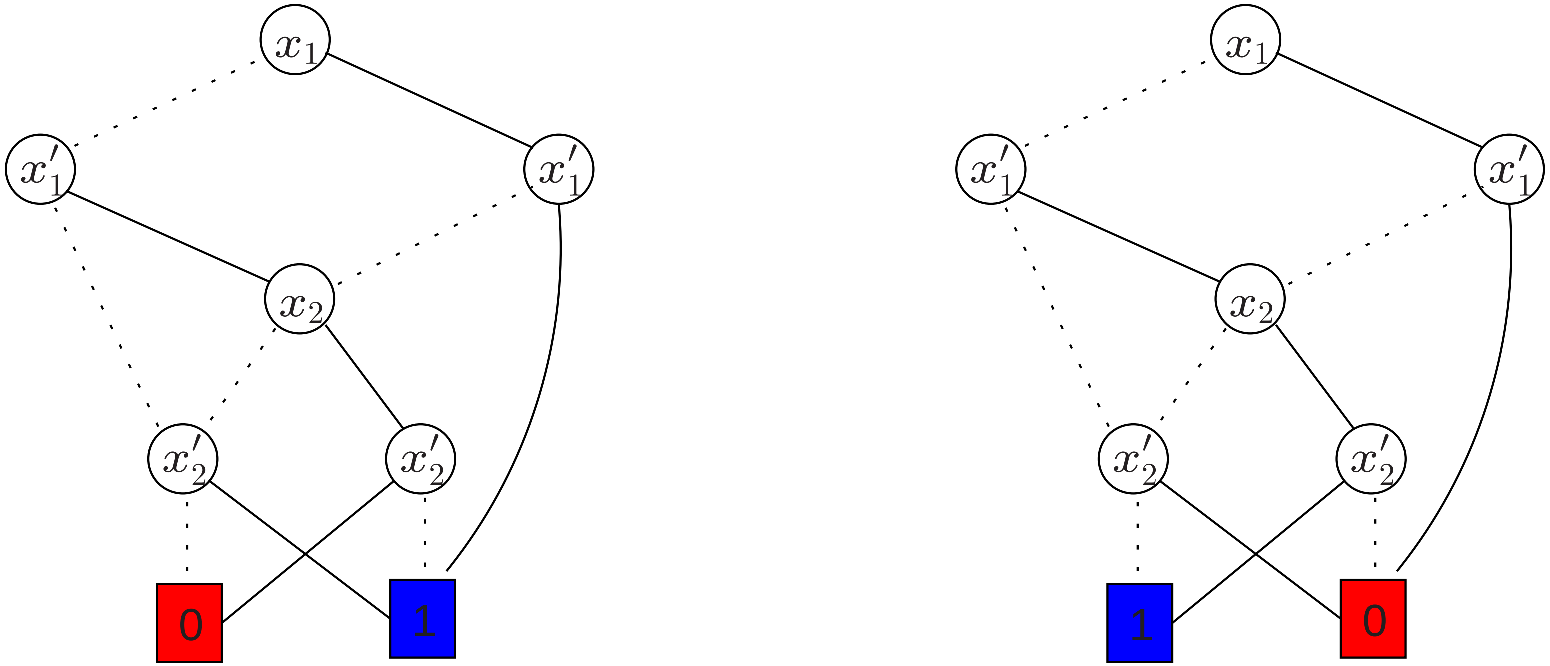
- This variable ordering yields compact ROBDDs for binary relations
  - for transition relation with $z_1 \ldots z_m$ be the encoding of action α, take:

$$\underbrace{z_1 < z_2 < \ldots < z_m}_{\text{encoding of } \alpha} < \underbrace{x_1 < y_1 < x_2 < y_2 < \ldots < x_n < y_n}_{\text{interleaved order of statea}}$$

Operations on ROBDDs

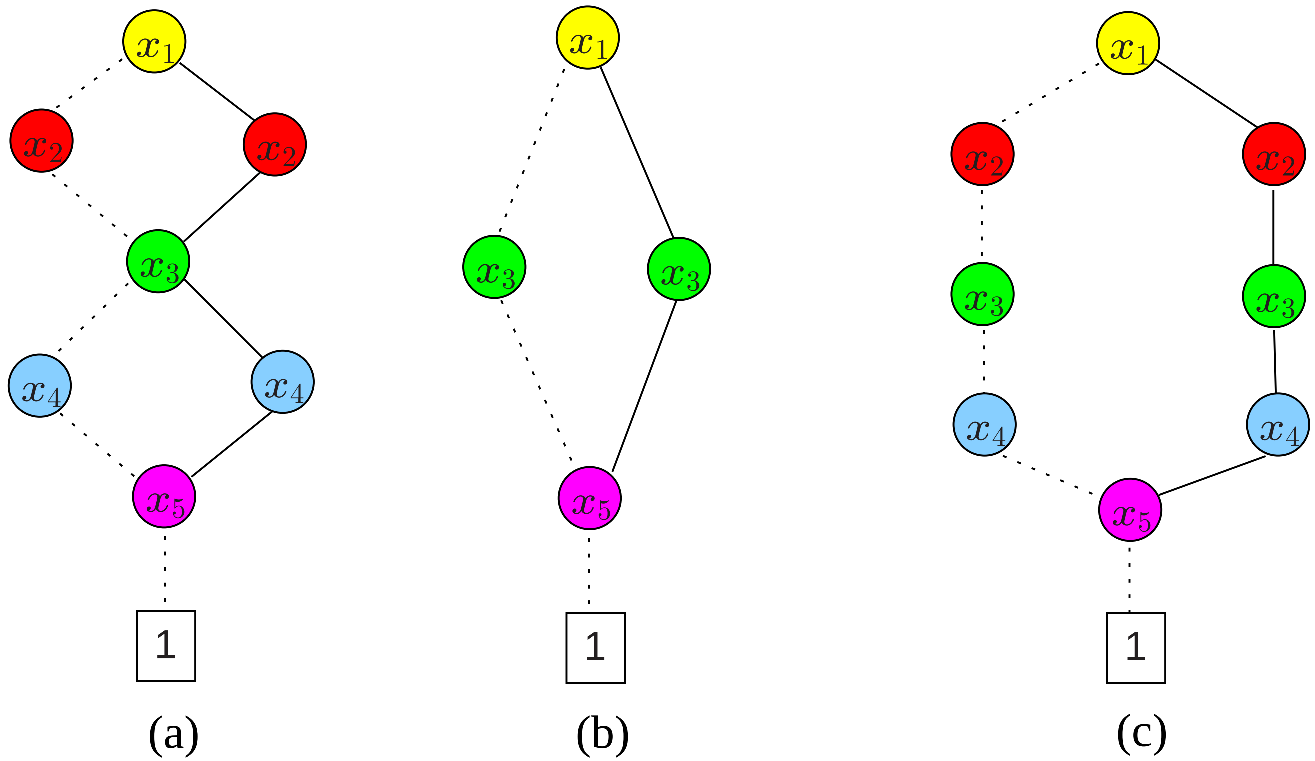
Algorithm	Inputs	Output ROBDD
REDUCE	B (not reduced)	B′ (reduced) with $f_{\mathsf{B}} = f_{\mathsf{B}'}$
NOT	B_f	$\mathsf{B}_{\neg f}$
APPLY	B_f, B_g, binary logical operator *op*	$\mathsf{B}_{f \textit{ op } g}$
RESTRICT	B_f, variable x, boolean value b	$\mathsf{B}_{f[x:=b]}$
RENAME	B_f, variables x and y	$\mathsf{B}_{f[x:=y]}$
EXISTS	B_f, variable x	$\mathsf{B}_{\exists x.\, f}$

Negation



negation amounts to interchange the 0- and 1-leaf

Conjunction



performing APPLY$(\wedge, \mathsf{B}_{left}, \mathsf{B}_{middle})$, i.e., compute $f_{\mathsf{B}_{left}} \wedge f_{\mathsf{B}_{middle}}$

APPLY

- Shannon expansion for binary operations:

$$\begin{aligned} f \text{ op } g \quad = \quad & (x_1 \ \wedge \ (f[x_1 := 1] \text{ op } g[x_1 := 1])) \\ \vee \ & (\neg x_1 \ \wedge \ (f[x_1 := 0] \text{ op } g[x_1 := 0])) \end{aligned}$$

- A top-down evaluation scheme using the Shannon's expansion:
  - let v be the variable highest in the ordering occurring in B_f orB_g
  - split the problem into subproblems for $v := 0$ and $v := 1$, and solve recursively
  - at the leaves, apply the boolean operator *op* directly
  - reduce afterwards to turn the resulting OBDD into an ROBDD
- Efficiency gain is obtained by *dynamic programming*
  - the time complexity of constructing the ROBDD of $\mathsf{B}_{f \text{ op } g}$ is in $\mathcal{O}(|\,\mathsf{B}_f\,| \cdot |\,\mathsf{B}_g\,|)$

Algorithm $\textsc{Apply}(op, \mathsf{B}_f, \mathsf{B}_g)$

B.root := $\textsc{Apply}(op, \mathsf{B}_f.\text{root}, \mathsf{B}_g.\text{root})$;

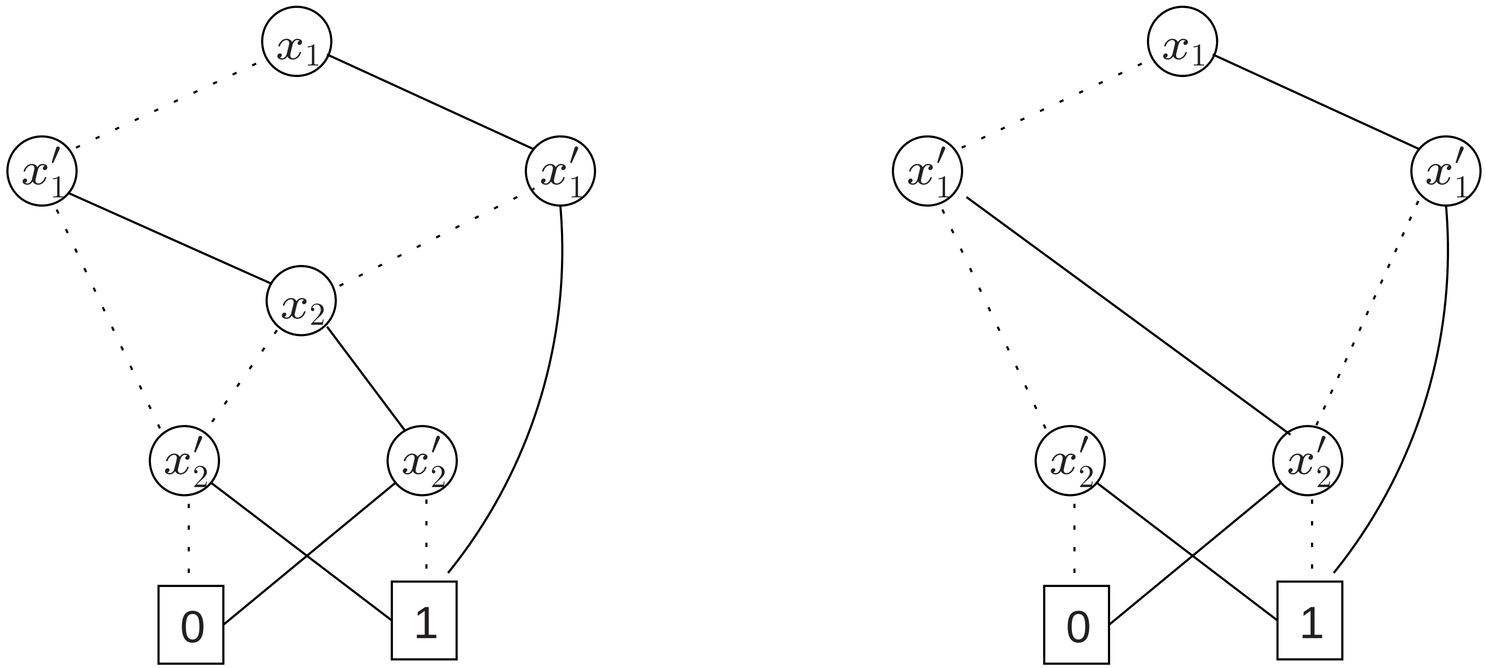
if $G(v_1, v_2) \neq$ empty **then return** $G(v_1, v_2)$ **fi**; (* lookup in hashtable *)
if (v_1 and v_2 are terminals) **then** *res* := *val*(v_1) *op val*(v_2) **fi**;
else if (v_1 is terminal and v_2 is nonterminal)
    then *res* := $MakeNode(Var(v_2), \textsc{Apply}(op, v_1, left(v_2)), \textsc{Apply}(op, v_1, right(v_2)))$;
else if (v_1 is nonterminal and v_2 is terminal)
    then *res* := $MakeNode(Var(v_1), \textsc{Apply}(op, left(v_1), v_2), \textsc{Apply}(op, right(v_1), v_2))$;
else if $(Var(v_1) = Var(v_2))$
    then *res* := $MakeNode(Var(v_1), \textsc{Apply}(op, left(v_1), left(v_2)), \textsc{Apply}(op, right(v_1), right(v_2)))$;
else if $(Var(v_1) < Var(v_2))$
    then *res* := $MakeNode(Var(v_1), \textsc{Apply}(op, left(v_1), v_2), \textsc{Apply}(op, right(v_1), v_2))$;
else (* $Var(v_1) > Var(v_2)$ *)
    res := $MakeNode(Var(v_2), \textsc{Apply}(op, v_1, left(v_2)), \textsc{Apply}(op, v_1, right(v_2)))$;
$G(v_1, v_2)$:= *res*; (* memoize result *)
return *res*

Example

Algorithm $\textsc{Restrict}(\mathsf{B}, x, b)$

- For each vertex v labeled with variable x:
  - if $b = 1$ then redirect incoming edges to *right*(v)
  - if $b = 0$ then redirect incoming edges to *left*(v)
  - remove vertex v, and (if necessary) reduce (only above v)

RESTRICT



performing RESTRICT(B, x_2, 1): replace x_2 by constant 1

EXISTS

- Existential quantification over x_i:

$$\exists x_i.\, f(x_1, \ldots, x_n) = f[x_i := 1] \;\vee\; f[x_i := 0]$$

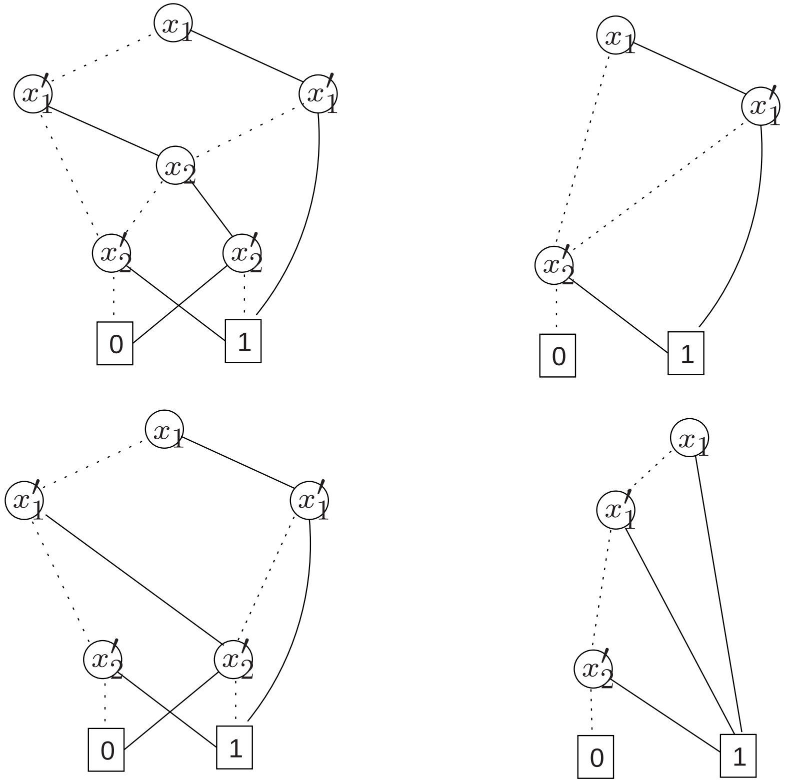
- Naive realization: $\text{APPLY}(\vee, \text{RESTRICT}(\mathrm{B}_f, x_i, 1), \text{RESTRICT}(\mathrm{B}_f, x_i, 0))$

- Efficiency gain:
  - observe that $\text{RESTRICT}(\mathrm{B}_f, x_i, 1)$ and $\text{RESTRICT}(\mathrm{B}_f, x_i, 0)$ are equal up to x_i
  - . . . the resulting ROBDD also has the same structure up to x_i
  - replace each node labeled with x_i by the result of applying $\vee$ on its children

- This can easily be generalized to $\exists x_1.\, \ldots\, \exists x_k.\, f(x_1, \ldots x_n)$

A simple example

A more involved example



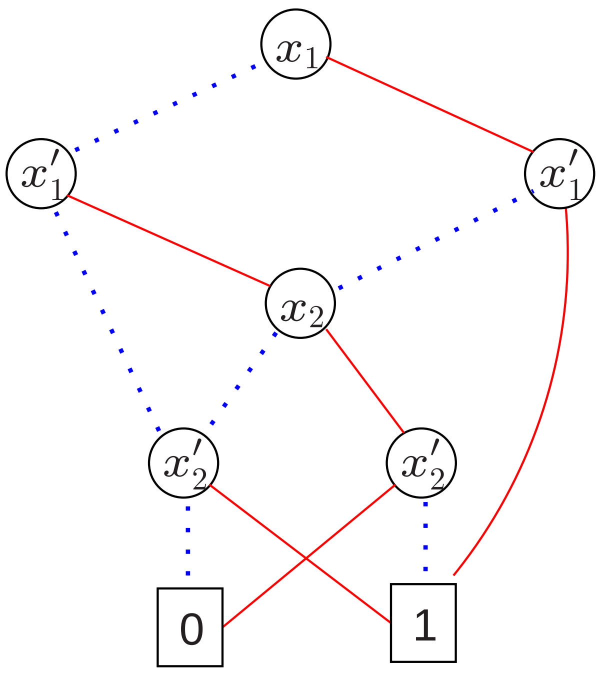
ROBBDs B_f (left up), $B_{f[x_2:=0]}$ (right up), $B_{f[x_2:=1]}$ (left down), and $B_{\exists x_2.\, f}$ (right down)

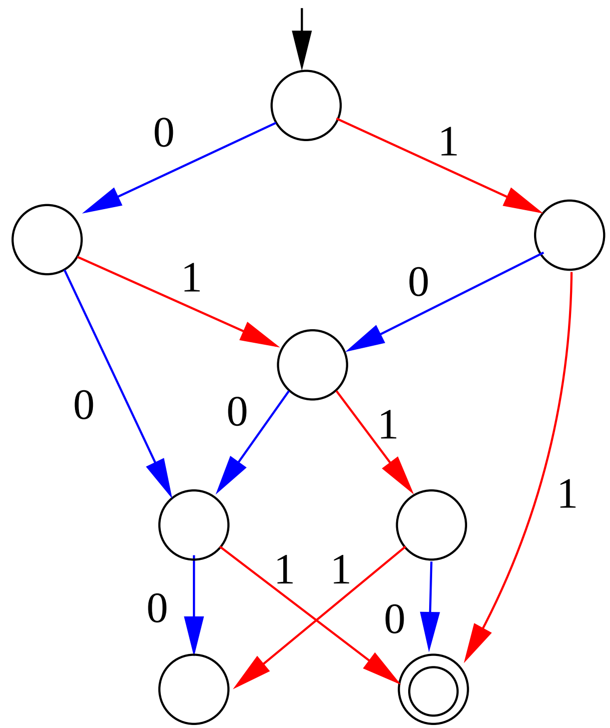
Operations on ROBDDs

Algorithm	Output	Time complexity	Space complexity
REDUCE	B′ (reduced) with $f_{\mathsf{B}} = f_{\mathsf{B}'}$	$\mathcal{O}(\vert B_f\vert \cdot \log \vert \mathsf{B}_f\vert)$	$\mathcal{O}(\vert \mathsf{B}_f\vert)$
NOT	$\mathsf{B}_{\neg f}$	$\mathcal{O}(\vert \mathsf{B}_f\vert)$	$\mathcal{O}(\vert \mathsf{B}_f\vert)$
APPLY	$\mathsf{B}_{f \textit{ op } g}$	$\mathcal{O}(\vert \mathsf{B}_f\vert \cdot \vert \mathsf{B}_g\vert)$	$\mathcal{O}(\vert \mathsf{B}_f\vert \cdot \vert \mathsf{B}_g\vert)$
RESTRICT	$\mathsf{B}_{f[x:=b]}$	$\mathcal{O}(\vert \mathsf{B}_f\vert)$	$\mathcal{O}(\vert \mathsf{B}_f\vert)$
RENAME	$\mathsf{B}_{f[x:=y]}$	$\mathcal{O}(\vert \mathsf{B}_f\vert)$	$\mathcal{O}(\vert \mathsf{B}_f\vert)$
EXISTS	$\mathsf{B}_{\exists x.\, f}$	$\mathcal{O}(\vert \mathsf{B}_f\vert^2)$	$\mathcal{O}(\vert \mathsf{B}_f\vert^2)$

operations are only efficient if f (and g) have compact ROBDD representations

OBDDs versus automata





each OBDD B *is a deterministic automaton* A_{B} *with* $f_{\mathsf{B}}^{-1}(1) = L(A_{\mathsf{B}})$

Analogies between ROBDDs and automata

- For language L, a minimised automaton is unique up to isomorphism
  - for a given variable ordering $<$, and function f, an ROBDD is unique upto $\cong$
- $L = L'$? can be checked by verifying isomorphism of their automata
  - $f = f'$? for boolean functions can be checked by verifying $\mathsf{B}_f \cong \mathsf{B}_{f'}$

$\Rightarrow$ in both cases, efficient algorithms do exist for this

- $L \neq \varnothing$? $\equiv$ is there a reachable accept state?
  - is f satisfiable? $\equiv$ its ROBDD has a reachable leaf 1
- Union, intersection, and complementation on automata is efficient
  - disjunction, conjunction, and negation on ROBDDs are efficient